

Análise Projeto de Algoritmos

Loana Tito Nogueira

16-Maio-2006

Projeto de Algoritmos por Divisão e Conquista

- **Dividir para Conquistar:** tática de guerra aplicada ao projeto de algoritmos
- Um algoritmo de divisão e conquista é aquele que resolve o problema desejado combinando as soluções parciais de (um ou mais) subproblemas, obtidas recursivamente.

Projeto de Algoritmos por Divisão e Conquista

- Seja H um problema algorítmico. O método da divisão e conquista consiste em tentar resolver H , através dos seguintes passos:
 - Decompor H em subproblemas $H_1, H_2, \dots, H_k$

Menores que H

Projeto de Algoritmos por Divisão e Conquista

- Seja H um problema algorítmico. O método da divisão e conquista consiste em tentar resolver H , através dos seguintes passos:
 - Decompor H em subproblemas $H_1, H_2, \dots, H_k$


Menores que H
 - Resolver cada subproblema H_i , através de uma aplicação recursiva desse método

Projeto de Algoritmos por Divisão e Conquista

- Seja H um problema algorítmico. O método da divisão e conquista consiste em tentar resolver H , através dos seguintes passos:
 - Decompor H em subproblemas $H_1, H_2, \dots, H_k$

Menores que H
 - Resolver cada subproblema H_i , através de uma aplicação recursiva desse método
 - Compor a solução de H , a partir das soluções de $H_1, H_2, \dots, H_k$

Algoritmo Genérico – Divisão e Conquista

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Entrada: A instância x

Saída: Solução y da instância x

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Entrada: A instância x

Saída: Solução y da instância x

Se x é suficientemente pequeno **então**

Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Entrada: A instância x

Saída: Solução y da instância x

Se x é suficientemente pequeno **então**

Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

Senão

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Entrada: A instância x

Saída: Solução y da instância x

Se x é suficientemente pequeno **então**

Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

Senão

Divisão

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Entrada: A instância x

Saída: Solução y da instância x

Se x é suficientemente pequeno **então**

Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

Senão

Divisão

Decomponha x em instâncias menores $x_1, \dots, x_k$

Para $i=1$ até k **faça** $y_i = \text{DivConq}(x_i)$

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Entrada: A instância x

Saída: Solução y da instância x

Se x é suficientemente pequeno **então**

Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

Senão

Divisão

Decomponha x em instâncias menores $x_1, \dots, x_k$

Para $i=1$ até k **faça** $y_i = \text{DivConq}(x_i)$

Conquista

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Entrada: A instância x

Saída: Solução y da instância x

Se x é suficientemente pequeno **então**

Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

Senão

Divisão

Decomponha x em instâncias menores $x_1, \dots, x_k$

Para $i=1$ até k **faça** $y_i = \text{DivConq}(x_i)$

Conquista

Combine as soluções y_i para obter a solução y de x

Algoritmo Genérico – Divisão e Conquista

- **DivConq(x)**

Entrada: A instância x

Saída: Solução y da instância x

Se x é suficientemente pequeno **então**

Retorne Solução(x) (dada pelo algoritmo para peq. Instâncias)

Senão

Divisão

Decomponha x em instâncias menores $x_1, \dots, x_k$

Para $i=1$ até k **faça** $y_i = \text{DivConq}(x_i)$

Conquista

Combine as soluções y_i para obter a solução y de x

Retorne(y)

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Caso base: $n=0$; $a^0 = 1$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n > 0$ e real a , sei calcular a^{n-1}

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n > 0$ e real a , sei calcular a^{n-1}

Passo da Indução: Queremos provar que conseguimos calcular a^n , para $n > 0$.

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

1a. Solução: (**Indução Fraca**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n > 0$ e real a , sei calcular a^{n-1}

Passo da Indução: Queremos provar que conseguimos calcular a^n , para $n > 0$.

Sei calcular a^{n-1}  calculo na multiplicando a^{n-1} por a

Algoritmo: Expon(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Se $n = 0$ então

 Retorne(1) {caso base}

Senão

$an' := \text{Expon}(a, n-1)$

$an := an' * a$

Retorne(an)

Algoritmo: Expon(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Se $n = 0$ então

Retorne(1) {caso base}

Senão

$an' := \text{Expon}(a, n-1)$

$an := an' * a$

Retorne(an)

Algoritmo: Expon(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Se $n = 0$ então

Retorne(1) {caso base}

Senão

$an' := \text{Expon}(a, n-1)$

$an := an' * a$

Retorne(an)

Algoritmo: Expon(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Se $n = 0$ então

 Retorne(1) {caso base}

Senão

$an' := \text{Expon}(a, n-1)$

$an := an' * a$

Retorne(an)

Complexidade: Expon(a,n)

- $T(n)$: n° de operações realizadas pelo algoritmo para calcular a^n

$$T(n) = \begin{cases} 1, & n=0 \\ T(n-1) + 1, & n>0 \end{cases}$$

Complexidade: Expon(a,n)

- $T(n)$: n° de operações realizadas pelo algoritmo para calcular a^n

$$T(n) = \begin{cases} 1, & n=0 \\ T(n-1) + 1, & n>0 \end{cases}$$



$$T(n) = \Theta(n)$$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: $n=0$; $a^0 = 1$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n > 0$ e real a , sei calcular a^k , para todo $k < n$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n > 0$ e real a , sei calcular a^k , para todo $k < n$

Passo da Indução: Queremos provar que conseguimos calcular a^n , para $n > 0$.

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**
 - **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n > 0$ e real a , sei calcular a^k , para todo $k < n$

Passo da Indução: Queremos provar que conseguimos calcular a^n , para $n > 0$.

Sei calcular $a^{\lfloor n/2 \rfloor}$

Projeto por Divisão e Conquista – Exemplo 1

- **Exponenciação**

- **Problema:** Calcular a^n , para todo real a e inteiro $n \geq 0$.

2a. Solução: (**Indução Forte**):

Caso base: $n=0$; $a^0 = 1$

Hipótese de Indução: Suponha que, para qualquer inteiro $n > 0$ e real a , sei calcular a^k , para todo $k < n$

Passo da Indução: Queremos calcular a^n para $n > 0$.

Sei calcular $a^{\lfloor n/2 \rfloor}$ →

$$\begin{cases} (a^{\lfloor n/2 \rfloor})^2, & \text{se } n \text{ é par} \\ a \cdot (a^{\lfloor n/2 \rfloor})^2, & \text{se } n \text{ é ímpar} \end{cases}$$

para

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Se $n = 0$ então

Retorne(1) {caso base}

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Se $n = 0$ **então**

Retorne(1) {caso base}

Senão

$an' := \text{Exp}(a, n \text{ div } 2)$

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Se $n = 0$ **então**

Retorne(1) {caso base}

Senão

 divisão

$an' := \text{Exp}(a, n \text{ div } 2)$

 conquista

$an := an' * an'$

Se $(n \bmod 2) = 1$ **então** $an := an * a$

Algoritmo: Exp(a,n)

Entrada: A base a e o expoente n

Saída: O valor de a^n

Se $n = 0$ **então**

Retorne(1) {caso base}

Senão

divisão

$an' := \text{Exp}(a, n \text{ div } 2)$

conquista

$an := an' * an'$

Se $(n \bmod 2) = 1$ **então** $an := an * a$

Retorne(an)

Complexidade

- $T(n)$: nº de operações realizadas pelo algoritmo para calcular a^n

$$T(n) = \begin{cases} 1 & , n=0 \\ T(\lfloor n/2 \rfloor) + c & , n>0 \end{cases}$$

Complexidade

- $T(n)$: nº de operações realizadas pelo algoritmo para calcular a^n

$$T(n) = \begin{cases} 1 & , n=0 \\ T(\lfloor n/2 \rfloor) + c & , n>0 \end{cases}$$



$$T(n) = \Theta(\log n)$$

Busca Binária

- **Problema:** Dado um vetor ordenado A com n números reais e um real x , determinar a posição $1 \leq i \leq n$ tal que $A[i]=x$, ou que não existe tal i .

Busca Binária

- **Problema:** Dado um vetor ordenado A com n números reais e um real x , determinar a posição $1 \leq i \leq n$ tal que $A[i]=x$, ou que não existe tal i .
- **INDUÇÃO FORTE!!**

Busca Binária

- **Problema:** Dado um vetor ordenado A com n números reais e um real x , determinar a posição $1 \leq i \leq n$ tal que $A[i]=x$, ou que não existe tal i .
- **INDUÇÃO FORTE!!**
 - Como o vetor está ordenado, conseguimos determinar, com apenas uma comparação, que metade das posições do vetor não podem conter o valor x

Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que $A[j] = x$ ou $j = 0$

Se $i=f$ então se $A[i] = x$ então retorne(i)

senão retorne(0)

senão

$j := (i+f) \text{ div } 2$

se $A[j] = x$ retorne(j)

senão se $A[j] > x$ então

$j := \text{BuscaBinaria}(A, i, f-1, x)$

senão

$j := \text{BuscaBinaria}(A, i+1, f, x)$

retorne(j)

Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que $A[j] = x$ ou $j = 0$

Se $i=f$ então se $A[i] = x$ então retorne(i)

senão retorne(0)

senão

$j := (i+f) \text{ div } 2$

se $A[j] = x$ retorne(j)

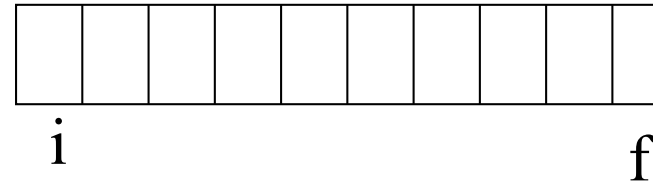
senão se $A[j] > x$ então

$j := \text{BuscaBinaria}(A, i, f-1, x)$

senão

$j := \text{BuscaBinaria}(A, i+1, f, x)$

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que $A[j] = x$ ou $j = 0$

Se $i=f$ então se $A[i] = x$ então retorne(i)

senão retorne(0)

senão

$j := (i+f) \text{ div } 2$

se $A[j] = x$ retorne(j)

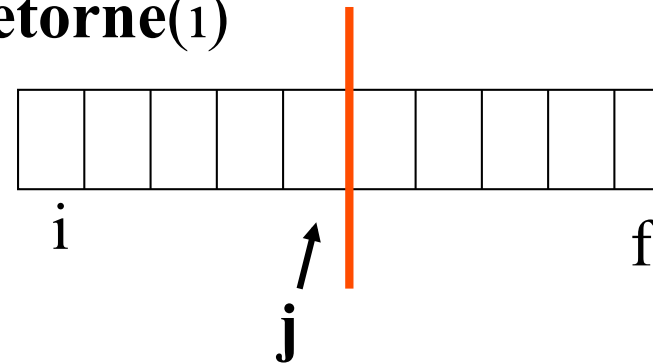
senão se $A[j] > x$ então

$j := \text{BuscaBinaria}(A, i, f-1, x)$

senão

$j := \text{BuscaBinaria}(A, i+1, f, x)$

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que $A[j] = x$ ou $j = 0$

Se $i=f$ então se $A[i] = x$ então retorne(i)

senão retorne(0)

senão

$j := (i+f) \text{ div } 2$

se $A[j] = x$ retorne(j)

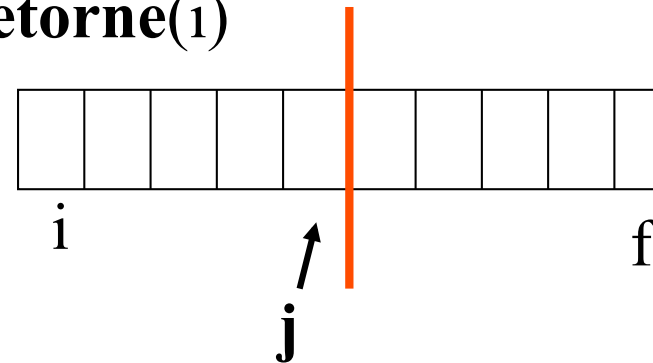
senão se $A[j] > x$ então

$j := \text{BuscaBinaria}(A, i, f-1, x)$

senão

$j := \text{BuscaBinaria}(A, i+1, f, x)$

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que $A[j] = x$ ou $j = 0$

Se $i=f$ então se $A[i] = x$ então retorne(i)

senão retorne(0)

senão

$j := (i+f) \text{ div } 2$

se $A[j] = x$ retorne(j)

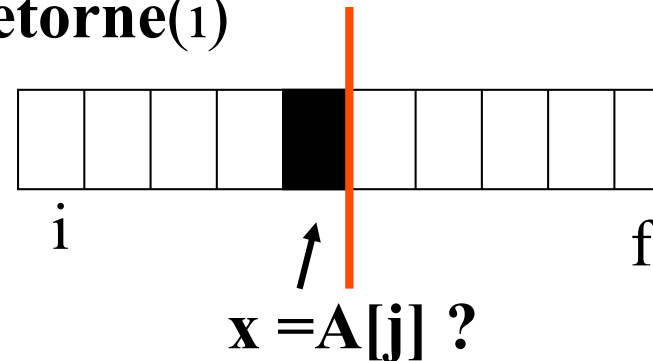
senão se $A[j] > x$ então

$j := \text{BuscaBinaria}(A, i, f-1, x)$

senão

$j := \text{BuscaBinaria}(A, i+1, f, x)$

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que $A[j] = x$ ou $j = 0$

Se $i=f$ então se $A[i] = x$ então retorne(i)

senão retorne(0)

senão

$j := (i+f) \text{ div } 2$

se $A[j] = x$ retorne(j)

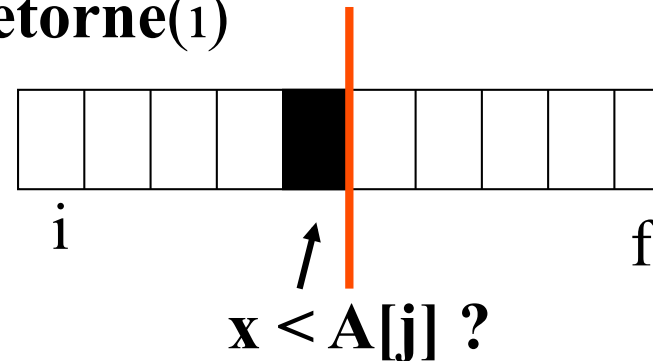
senão se $A[j] > x$ então

$j := \text{BuscaBinaria}(A, i, j-1, x)$

senão

$j := \text{BuscaBinaria}(A, i+1, f, x)$

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que $A[j] = x$ ou $j = 0$

Se $i=f$ então se $A[i] = x$ então retorne(i)

senão retorne(0)

senão

$j := (i+f) \text{ div } 2$

se $A[j] = x$ retorne(j)

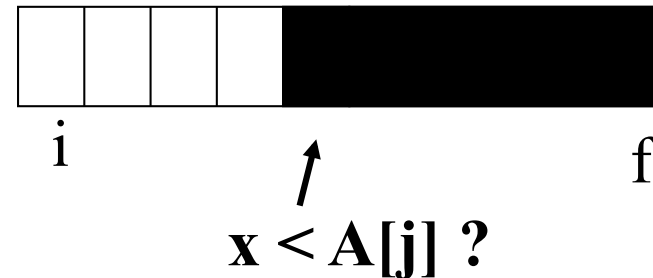
senão se $A[j] > x$ então

$j := \text{BuscaBinaria}(A, i, j-1, x)$

senão

$j := \text{BuscaBinaria}(A, i+1, f, x)$

retorne(j)



Algoritmo: BuscaBinária(A, i, f, x)

Entrada: Vetor A, extremos do subvetor, i e f e x

Saída: índice $1 \leq j \leq n$ tal que $A[j] = x$ ou $j=0$

Se $i=f$ então se $A[i] = x$ então retorne(i)

senão retorne(0)

senão

$j := (i+f) \text{ div } 2$

se $A[j] = x$ retorne(j)

senão se $A[j] > x$ então

$j := \text{BuscaBinaria}(A, i, f-1, x)$

senão

$j := \text{BuscaBinaria}(A, i+1, f, x)$

retorne(j)

Complexidade

- CALCULE!

Ordenação

- **Problema:** Ordenar um conjunto de $n \geq 1$ inteiros
- Diversos algoritmos para o problema de ordenação através de indução

Ordenação

- **Indução Simples:**

- **Hipótese de Indução:**

Sabemos ordenar um conjunto de $n-1 \geq 1$ inteiros

Ordenação

- **Indução Simples:**
 - **Hipótese de Indução:**
Sabemos ordenar um conjunto de $n-1 \geq 1$ inteiros

Caso base: $n=1$. Já está ordenado!!

Ordenação

- **Indução Simples:**

- **Hipótese de Indução:**

Sabemos ordenar um conjunto de $n-1 \geq 1$ inteiros

Caso base: $n=1$. Já está ordenado!!

Passo da Indução: Seja S um conjunto com $n > 1$ inteiros e x um elemento de S .

Ordenação

- **Indução Simples:**

- **Hipótese de Indução:**

Sabemos ordenar um conjunto de $n-1 \geq 1$ inteiros

Caso base: $n=1$. Já está ordenado!!

Passo da Indução: Seja S um conjunto com $n > 1$ inteiros e x um elemento de S .

Por h.i. sabemos ordenar $S-x$, basta então inserir x na posição correta para obtermos S ordenado

Ordenação

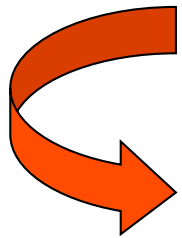
- **Indução Simples:**

- **Hipótese de Indução:**

Sabemos ordenar um conjunto de $n-1 \geq 1$ inteiros

Caso base: $n=1$. Já está ordenado!!

Passo da Indução: Seja S um conjunto com $n > 1$ inteiros e x um elemento de S .



Por h.i. sabemos ordenar $S-x$, basta então inserir x na posição correta para obtermos S ordenado

INSERTION SORT!!!

Algoritmo: OrdenaçãoInserção(A, n)

Entrada: Vetor A e n inteiro

Saída: Vetor A ordenado

Se $n \geq 2$ faça

OrdenaçãoInserção(A, n-1)

$v := A[n]$

$j := n$

enquanto $(j > 1)$ e $(A[j-1]) > v$ faça

$A[j] := A[j-1]$

$j := j-1$

$A[j] := v$

Algoritmo: OrdenaçãoInserção(A, n)

Entrada: Vetor A e n inteiro

Saída: Vetor A ordenado

Se $n \geq 2$ faça

OrdenaçãoInserção(A, n-1)

$v := A[n]$

$j := n$

enquanto $(j > 1)$ e $(A[j-1]) > v$ faça

$A[j] := A[j-1]$

$j := j-1$

$A[j] := v$

Versão Recursiva

Complexidade

- Tanto o número de comparações quanto de trocas é dado pela recorrência:

- $$T(n) = \begin{cases} 0, & n=1 \\ T(n-1) + n, & n>1 \end{cases}$$

Algoritmo: OrdenaçãoInserção(A, n)

Entrada: Vetor A e n inteiro

Saída: Vetor A ordenado

Para $i:=2$ até n **faça**

$v:= A[n]$

$j:= n$

enquanto $(j > 1)$ e $(A[j-1]) > v$ **faça**

$A[j]:=A[j-1]$

$j:=j-1$

$A[j]:=v$

Versão Iterativa

2a. alternativa

- **Hipótese de Indução:**

Sabemos ordenar um conjunto de $n-1 \geq 1$ inteiros

Caso base: OK!

Passo da Indução: Encontrar o mínimo x .

Sabemos ordenar $S \setminus x$.



Selection Sort

Selection Sort(A, i, n)

Se $i < n$ **faça**

$\text{min} := i$

para $j := i+1$ **até** n **faça**

se $A[j] < A[\text{min}]$ **então** $\text{min} := j$

$t := A[\text{min}]$

$A[\text{min}] := A[i]$

$A[i] := t$

SelectionSort($a, i+1, n$)

Entrada: Vetor A, início e fim

Saída: Vetor A ordenado

Complexidade

- Número de comparações:
 - $T(n) = 0$, se $n=1$
 - $T(n) = T(n-1) + n$, se $n > 1$
- Número de trocas:
 - $T(n) = 0$, se $n=1$
 - $T(n) = T(n-1) + 1$, $n > 1$

Complexidade

- Número de comparações:

- $T(n) = 0$, se $n=1$
- $T(n) = T(n-1) + n$, se $n > 1$

$$\Theta(n^2)$$

- Número de trocas:

- $T(n) = 0$, se $n=1$
- $T(n) = T(n-1) + 1$, $n > 1$

$$\Theta(n)$$

Versão Iterativa

Para $i:=1$ **até** $n-1$ **faça**

$\text{min}:=i$

para $j:=i+1$ **até** n **faça**

se $A[j] < A[\text{min}]$ **então** $\text{min}:=j$

$t:=A[\text{min}]$

$A[\text{min}]:=A[i]$

$A[i]:=t$

Indução Forte

- **Hipótese de Indução:**

Sabemos ordenar um conjunto de $1 \leq k \leq n$

Indução Forte

- **Hipótese de Indução:**
Sabemos ordenar um conjunto de $1 \leq k \leq n$
- **Caso Base:** $n=1$. OK!
- **Passo da Indução:** Seja S um conjunto com $n > 1$ inteiros.
Podemos particionar S em dois subconjuntos de tamanhos aproximadamente iguais: $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

Indução Forte

- **Hipótese de Indução:**

Sabemos ordenar um conjunto de $1 \leq k \leq n$

- **Caso Base:** $n=1$. OK!

- **Passo da Indução:** Seja S um conjunto com $n > 1$ inteiros.

Podemos particionar S em dois subconjuntos de tamanhos aproximadamente iguais: $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

$< n$

Indução Forte

- **Hipótese de Indução:**

Sabemos ordenar um conjunto de $1 \leq k \leq n$

- **Caso Base:** $n=1$. OK!

- **Passo da Indução:** Seja S um conjunto com $n > 1$ inteiros. Podemos particionar S em dois subconjuntos de tamanhos aproximadamente iguais: $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

→ Podemos usar a h.i. para ordenar S_1 e S_2

Indução Forte

- **Hipótese de Indução:**
Sabemos ordenar um conjunto de $1 \leq k \leq n$
- **Caso Base:** $n=1$. OK!
- **Passo da Indução:** Seja S um conjunto com $n > 1$ inteiros.
Podemos particionar S em dois subconjuntos de tamanhos aproximadamente iguais: $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

→ Podemos usar a h.i. para ordenar S_1 e S_2

COMO OBTER S ?

Algoritmo:

- O algoritmo emprega os seguintes procedimentos:

Algoritmo:

- O algoritmo emprega os seguintes procedimentos:
 - SORT(i,j): ordena o subconjunto $\{s_i, \dots, s_j\}$

Algoritmo:

- O algoritmo emprega os seguintes procedimentos:
 - $\text{SORT}(i,j)$: ordena o subconjunto $\{s_i, \dots, s_j\}$
 - $\text{MERGE}(S_1, S_2)$: efetua a intercalação dos subconjuntos ordenados S_1 e S_2

Algoritmo:

- O algoritmo emprega os seguintes procedimentos:
 - $\text{SORT}(i,j)$: ordena o subconjunto $\{s_i, \dots, s_j\}$
 - $\text{MERGE}(S_1, S_2)$: efetua a intercalação dos subconjuntos ordenados S_1 e S_2
 - Chamada Externa: **$\text{SORT}(1,n)$**

Algoritmo

Procedimento SORT(i, j)

Se $i=j$ então

retornar s_i

Senão

$m := (i+j-1)/2$

Retornar MERGE(SORT(i, m), SORT($m+1, j$))

Algoritmo

Procedimento MERGE(S_1, S_2)

Se $S_1 = \emptyset$ **então**

Retornar S_2

senão

se $S_2 = \emptyset$ **então**

Retornar S_1

senão

 sejam e_1, e_2 os elementos iniciais de S_1 e S_2 resp.

$k :=$ **Se** $e_1 > e_2$ **então** 1 **senão** 2

retornar $\{e_k\}$

$S_k := S_k - \{e_k\}$

 MERGE(S_1, S_2)

Complexidade

- $T(n) = 2T(n/2) + n - 1$

